

# Docker

---

## Differences between virtualisation and containerisation

Feature	Virtualization 	Containerization 
Definition	Runs multiple OSes on a hypervisor.	Runs lightweight apps using shared OS.
Isolation	Each VM has its own OS.	Containers share the host OS.
Resource Usage	High (each VM includes a full OS).	Low (containers share the OS kernel).
Performance	Slower (full OS overhead).	Faster (directly runs on the host OS).
Startup Time	Minutes (loads entire OS).	Seconds (only starts the app).
Portability	Limited (large VM images).	Highly portable (runs anywhere).
Use Case	Running multiple OS environments.	Microservices, cloud apps, CI/CD.
Tools	VMware, VirtualBox, Hyper-V.	Docker, Kubernetes, Podman.

## What's Included in a Container vs. Virtual Machine?

Component	Container 	Virtual Machine 
Kernel	Shared with the host OS	Separate for each VM
Operating System	Not included (uses host OS)	Full OS (Windows, Linux, etc.)
Libraries & Dependencies	Included for the app only	Included with the full OS
Application	Runs inside the container	Runs inside the VM
Hypervisor	 Not needed	 Required to manage VMs

## Benefits of Each (Virtual Machines vs. Containers)

### Virtual Machines (VMs) Benefits

#### Full Isolation:

Each VM has its own OS, providing strong security. Ideal for running different OSes on one machine (e.g., Windows and Linux).

#### Better for Legacy Apps:

Great for apps that require a specific OS or system dependencies.

#### Stable & Reliable:

Works well in traditional enterprise environments.

#### Supports GUI-based Applications:

Unlike containers, VMs can run full desktop environments.

### Container Benefits

Lightweight & Fast:

Starts in seconds (no OS boot required).

Uses fewer resources since multiple containers share the same OS.

 Highly Portable:

"Build once, run anywhere" (works the same across different environments).

 Better for Microservices & Cloud:

Works well for DevOps, CI/CD, and modern cloud applications.

 Scalability & Efficiency:

Uses less memory and CPU compared to VMs, allowing more apps per server.

## Microservices

### What are they?

Microservices are a software architecture style where an application is broken down into small, independent services that communicate with each other. Each microservice:

- ✓ Handles a specific function (e.g., authentication, payment, inventory).
- ✓ Communicates via APIs (usually REST or gRPC).
- ✓ Can be developed, deployed, and scaled independently.

### How Are Microservices Made Possible?

#### 1. Containers & Orchestration

- Docker: Packages microservices into isolated, portable containers.
- Kubernetes: Manages, scales, and orchestrates microservices across multiple servers.

#### 2. API Communication

- RESTful APIs or gRPC: Allows microservices to talk to each other.
- Message Brokers (Kafka, RabbitMQ): Helps in asynchronous communication.

#### 3. DevOps & CI/CD

- Continuous Integration/Continuous Deployment (CI/CD): Automates testing and deployment.
- Infrastructure as Code (IaC) (Terraform, Ansible): Automates cloud/server provisioning.

#### 4. Cloud & Scalability

- Cloud Services (AWS, Azure, GCP): Provides managed services to deploy microservices.
- Service Discovery (Eureka, Consul): Helps services locate each other dynamically.

## Docker- What is it

### What is it?

Docker is a containerization platform that allows developers to package applications and their dependencies into lightweight, portable containers. These containers can run on any environment (local, cloud, or on-premise) without compatibility issues.

#### ◊ Key Features:

- Portability – "Write once, run anywhere"
- Lightweight – Uses fewer resources than virtual machines

- Fast Deployment – Containers start in seconds
- Scalability – Easily scale applications across multiple servers

## Alternatives

- Podman
- LXC (Linux Containers)
- Singularity

## Success story: Netflix

Netflix needed to:

Scale services rapidly across different cloud environments.

Improve developer productivity.

Reduce deployment time for thousands of microservices.

◊ The Docker Solution:

- Microservices with Docker: Netflix broke down its monolithic app into microservices, each running in Docker containers.
- Consistent Environment: Developers could create identical local environments to production using Docker.
- Scalability with Kubernetes: Containers were managed and scaled dynamically.

## Task: Run and pull your first image

- Command to see docker images already on your local machine: `docker image ls`
- Run command 'hello world' is: `docker run hello-world`
- First run: Docker checks if the image exists on your local machine.  
Since the image is missing, Docker pulls it from Docker Hub.
- Second run: Docker checks if the hello-world image is already on your machine. Since it's already downloaded, Docker does not pull it again.

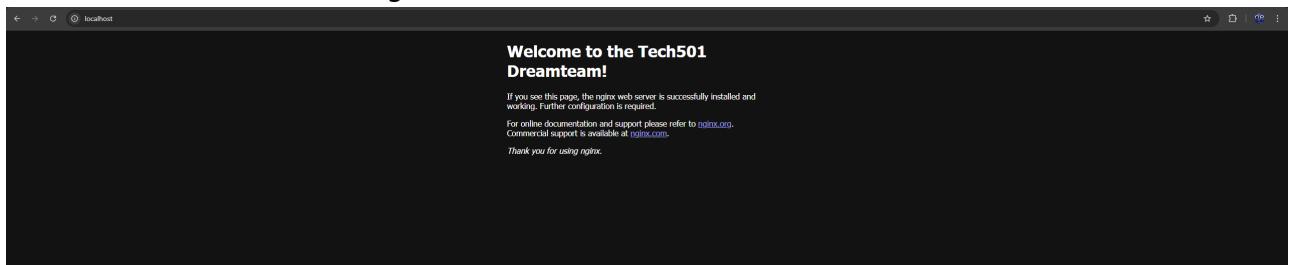
## Task: Run Nginx web server in a Docker container

`docker pull nginx:latest` `docker run -d -p 80:80 --name my-nginx nginx` - Creates the nginx and specifying the port for it to run on `docker start my-nginx`- to start nginx `docker ps` - to check if its running `docker ps -a` - to see all the containers `http://localhost` - to check if its working `docker stop my-nginx` - to stop the container running

## Task: Remove a container

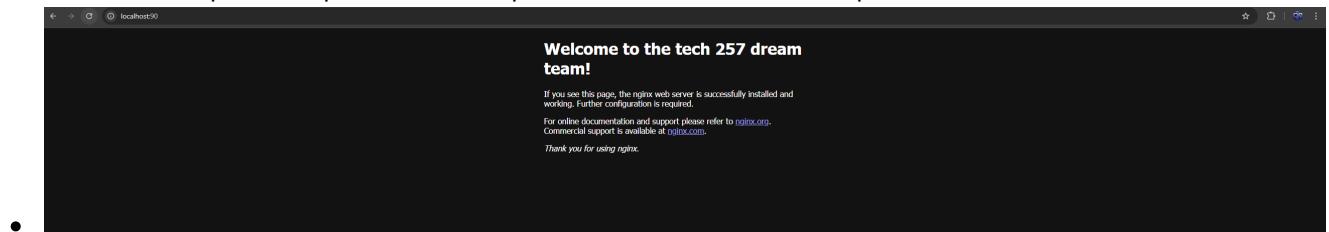
- `docker rm my-nginx` - To remove the nginx container. Doing whilst running will give you an error of:  
`Error response from daemon: cannot remove container "/my-nginx": container is running: stop the container before removing or force remove`
- `docker exec -it <container_name_or_id> /bin/bash`- the command to access the shell of a container, for my nginx it would be `docker exec -it my-nginx /bin/bash` - used windows powershell as it didn't work in git bash
- `apt-get update apt-get install sudo` - used to install sudo so we can run the update and upgrade commands
- `cd /usr/share/nginx/html` - command to get to the part where the index file is for the nginx.

- `apt-get update apt-get install nano` - To install nano command. `nano index.html` - to go into the index file to make the change



## Task: Run a different container on a different port

- Using the command: `docker run -d -p 80:80 --name dreamteam-nginx daraymonsta/nginx-257:dreamteam` - gives an error can't expose it to port 80 as it is already in use.
- `docker rm dreamteam-nginx` - to remove the container]
- `docker run -d -p 90:80 --name dreamteam-nginx daraymonsta/nginx-257:dreamteam` - command to put it on port 90, use `http://localhost:90` to see it on port 90



## Task: Push host-custom-static-webpage container image to Docker Hub

### Step 1: Get the container id

- Get the container id first by running the command `docker ps`

### Step 2: Commit the file

- Using this command `docker commit <container_id> yourdockerhubusername/custom-nginx`

### Step 3: Login your docker account in the terminal

- `docker login` - will tell you match the authorisation code from the terminal to the one in the browser

### Step 4: Push the image to your docker account

- `docker push myusername/custom-nginx:latest` -pushes it to your docker account

Docker command that runs the container which uses the pushed container image

- `docker run -d -p 80:80 --name my-nginx-container yourdockerhubusername/custom-nginx:latest`

## Task: Automate docker image creation using a Dockerfile

### step 1 - create docker file

## ⚡ Dockerfile > ...

```

1   FROM nginx:1.21.6
2
3   WORKDIR /usr/share/nginx/html
4   COPY index.html .
5   # Expose port 80 for the Nginx web server
6   EXPOSE 80
7
8   CMD ["nginx", "-g", "daemon off;"]

```

## step 2 - create an index.html file

```

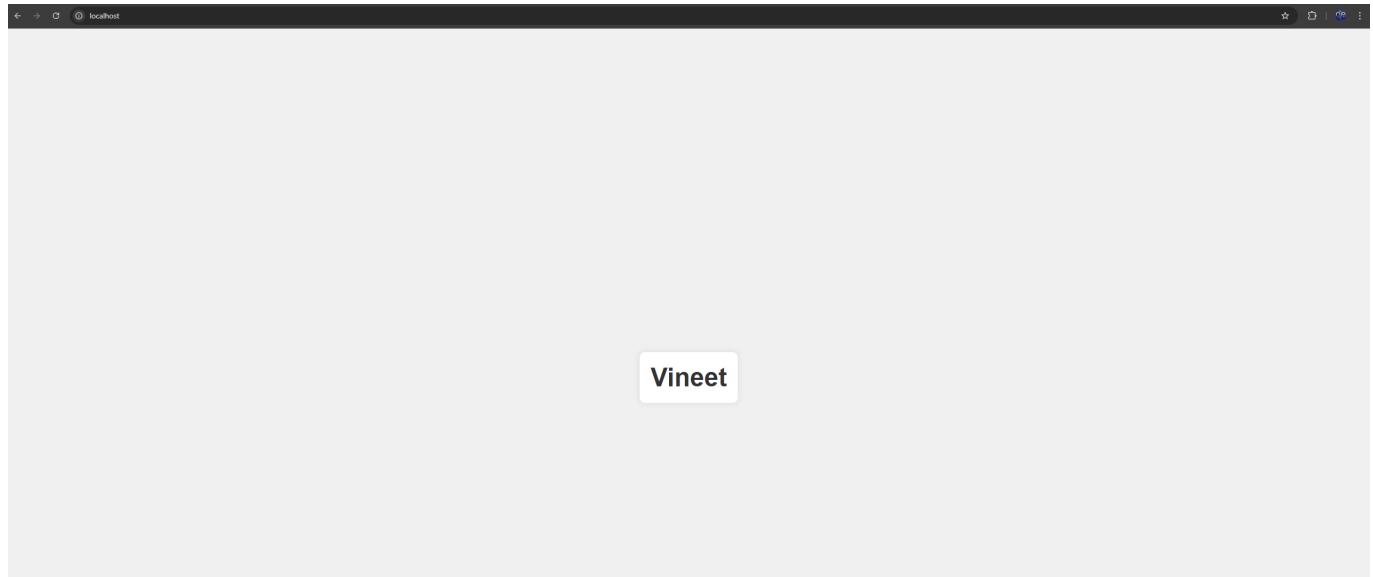
↳ index.html > ↗ html
1   <!DOCTYPE html>
2   <html lang="en">
3     <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>Vineet's Page</title>
7       <style>
8         body {
9           font-family: Arial, sans-serif;
10          background-color: #f0f0f0;
11          display: flex;
12          justify-content: center;
13          align-items: center;
14          height: 100vh;
15          margin: 0;
16        }
17        .container {
18          text-align: center;
19          padding: 20px;
20          background-color: white;
21          border-radius: 10px;
22          box-shadow: 0 0 10px rgba(0,0,0,0.1);
23        }
24        h1 {
25          color: #333;

```

## Step 3

- docker build -t tech501-mod-nginx-dockerfile . - creates the image
- docker run -d -p 80:80 --name my-nginx-container2 sha2 - creates the container
- docker rmi <image-id> to remove the image off your local
- docker run --pull always -d -p 80:80 --name my-nginx-container yourdockerhubusername/nginx-custom:latest

## Step 4: Run the localhost in the browser



Task: Run Sparta test app in a docker container

Step 1: Create a new repo and folder

Step 2: Move the app folder into the new folder

Step 3: Create a new Dockerfile in the folder

## Dockertile > ...

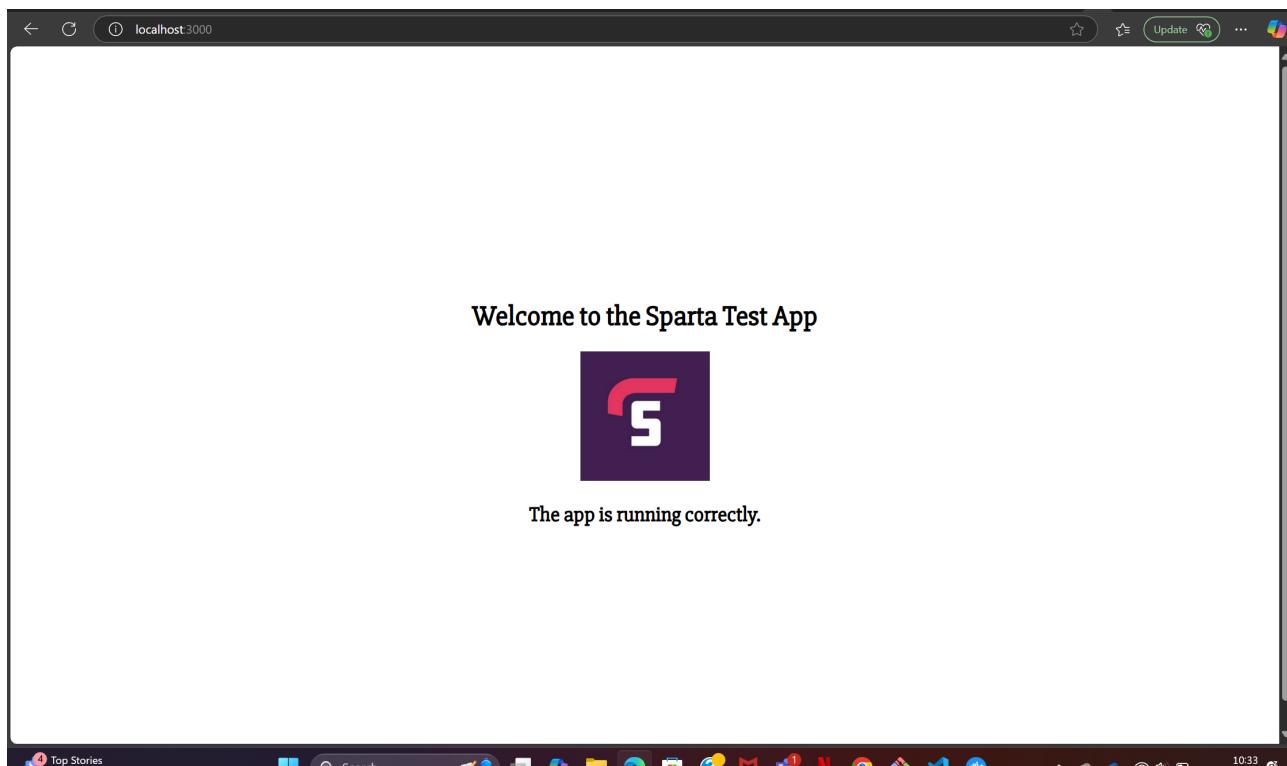
```
1  # 1 Base Image
2  FROM node:18
3
4  # 3 Set the Default Working Directory
5  WORKDIR /usr/src/app
6
7  # 4 Copy Application Files
8  COPY app /usr/src/app
9  COPY package*.json ./
10
11 # 5 Install Dependencies
12 RUN npm install
13
14 # 6 Expose the Port
15 EXPOSE 3000
16
17 # 7 Default Command to Start the App
18 CMD ["node", "app.js"]
19 # If your app uses `npm start`, replace with:
20 # CMD ["npm", "start"]
21
```

## Step 4: Build the docker image

- `docker build -t yourdockerhubusername/my-app:latest .` - this command will build the docker image

## Step 5: Run the docker container

- `docker run -d -p 3000:3000 --name my-running-app yourdockerhubusername/my-app:latest` - this command will run the docker container



## Step 6: Push the image to docker hub

- Login into docker account
- Use this command to push to docker account `docker push yourdockerhubusername/my-app:latest`

## Step 6: Force docker to pull a fresh image from docker hub

- First force the image to be removed from local
- `docker run --pull always -d -p 3000:3000 --name my-running-app yourdockerhubusername/my-app:latest`

## Task: Research Docker compose

### Why use it?

- Docker Compose is useful when you need to define and manage multi-container Docker applications. Instead of manually running multiple docker run commands, Compose allows you to define your entire application stack in a simple YAML file (docker-compose.yml). Here's why you should use Docker Compose:
  - Easy Configuration and Version Control
  - Simplifies Multi-Container Management
  - Consistency Across Environments
  - Dependency Management

### How to use it?

### What to install?

- You need to install docker compose

## How to store the compose file?

- A docker-compose.yml file should be stored in your project directory, typically at the root.

## Command to manage your application

- Start the Application (Without Detached Mode) = `docker compose up`
- Start the Application (In Detached Mode) = `docker compose up -d`
- The difference between with/without detached node: Detached (-d) -Runs in the background, keeps running even if you close the terminal. Foreground (default) Shows logs in the terminal, stops when you press CTRL + C.
- Stop the application: `docker compose down` or if you want to stop without removing the containers: `docker compose stop`
- Run application in detached mode= `docker compose up -d`
- Check Services Running with Docker Compose= `docker compose ps`
- View logs in real time= `docker compose logs -f`
- View docker compose images= `docker compose images`

## Task: Use Docker compose to run app and database containers

### Step 1: Pull the docker MongoDB image

- `docker pull mongo:6.0`
- Run it as a container `docker run -d \ --name my-mongodb \ -p 27017:27017 \ -v mongo-data:/data/db \ -e MONGO_INITDB_ROOT_USERNAME=admin \ -e MONGO_INITDB_ROOT_PASSWORD=password \ mongo:6.0`

### Step 2: Create a docker-compose.yml file

- `touch docker-compose.yml`
- `nano docker-compose.yml`

```

❶ docker-compose.yml
 1  version: "3.8"
 2
 3  services:
 4    app:
 5      image: vineetsethi1/my-app:latest # Replace with your actual image
 6      container_name: my-app
 7      restart: always
 8      command: sh -c "node seeds/seed.js && npm start"
 9      networks:
10        - app-network
11      ports:
12        - "3000:3000"
13      depends_on:
14        - mongodb
15      environment:
16        DB_HOST: "mongodb://mongodb:27017/mydatabase" # Connect app to MongoDB
17        #SEED_MONGODB: "true" # Seed MongoDB with sample data
18
19    mongodb:
20      image: mongo:6.0
21      container_name: my-mongodb
22      restart: always
23      networks:
24        - app-network
25      ports:
26        - "27017:27017"
27
28    networks:
29      app-network:
30        driver: bridge
31

```

### Step 3: Start the containers

- `docker compose up -d`

### Step 4: SSH into the app container

- `docker exec -it my-app sh`
- Create the DB\_HOST env variable

### Step 5: Make sure to keep the app and db in the same network

- This can be done in the docker compose file by specifying network in both the app and db code

### Step 6: You can either manually seed the database or automatically

- Manual:

- Go into the app container and run `npm install`

```
> docker exec -it my-app sh
> npm install

> sparta-test-app@1.0.1 postinstall
> node seeds/seed.js

Connected to database
Database cleared
Database seeded with 100 records
Database connection closed

Up to date, audited 389 packages in 7s

2 packages are looking for funding
  run 'npm fund' for details

2 vulnerabilities (3 low, 3 moderate, 5 high, 1 critical)

To address all issues, run:
  npm audit fix

Run 'npm audit' for details.
•
```

**Recent Posts**

---

**Pataca bandwidth**

Quo ratione corrupti saepe repellat optio nisi dolorem enim. Ea autem ex earum inventore sapiente et. Et voluptatibus dolor non blanditiis modi. Placeat et repudianda alias minus cumque qui est. Possimus et ullam dolor reprehenderit culpa. Non et sed enim quasi rerum incident corporis maxime. Fuga qui sit laudantium nam atque ut. Est incident eos nulla est accusantium. Autem fuga dolorem incident atque culpa. Aut beatae et veniam natus vero. Magnam atque aut occaecati quia consequatur molestiae dolorem minus vel. Dolorum explicabo quia aut quis ab doloremque ad. Harum blanditiis minima nemo minus dolorum. Neque laboriosam qui esse sequi vero et in.

**Oregon deposit**

Et omnis sed non. Quidem alias eos et et rerum accusamus ex perferendis aut. Vero ut dolor quia aliquam molestias omnis earum quis. Sed illum deserunt est aut nemo est cum officiis est. Ea aut velit repellat provident voluptatem. Deserunt occaecati quasi quibusdam nemo aut est autem. Tempore asperiores molestiae dolor atque voluptas. Voluptatibus velit delectus officiis est ut porro et ullam aperiam. Quos doloremque nostrum fuga qui harum. Officiis eveniet corporis natus ad sit nihil alias iste. Id quis consequatur eligendi odio ratione. Illum cupiditate minus ratione quod laudantium et.

**Texas asymmetric**

Facere voluptas sint qui quo. Voluptate velit et accusantium vel neque consequuntur. Aliquid asperiores deleniti consequatur expedita. Tempore laudantium voluptatem non et et modi delectus enim. Incident consequatur et magnam ab odit accusantium. Eaque aut explicabo perspicatis cupiditate aut numquam sunt perspicatis. Repellat repellat natus odit. Aperiam at consequuntur. Facere voluptates eius est qui a at voluptate ipsam id. Blanditiis adipisci est quis excepturi. Ut dolorem quis delectus ex aspernatur. Odio voluptatem eos. Vero natus voluptatem omnis eos laudantium. Rerum illo reiciens ipsa dolore non laborum id illum.

- 
- Automatic:

- Go into the compose file and a command **command: sh -c "node seeds/seed.js && npm start"**



## Recent Posts

---

### deliverables

Ipsum non numquam qui corrupti velit earum facilis ullam id. Eveniet optio sint aliquid vitae ipsam. Harum illo omnis quas. Non mollitia aliquam rerum autem sed ratione. Tempore consequuntur quod. Natus quis quis. Segui architecto ut eos aperiam illum perspicatis eum. Nobis et neque nisi illo ab repellendus perspicatis tempore quidem. Maxime et quibusdam deleniti sit quae quae molestias voluptates qui. Quia quo repudiandae voluptas error nihil expedita qui aut. Sunt nemo in est incident placet quia aperiam eum sed. Unde molestias ab ipsa ipsum. Possimus voluptatibus impedit qui ab dolorem praesentium. Tempore facere dolor est.

### Ukraine

Nam unde consequatur aliquid eius aut deserunt nam impedit qui. Repellendus quia nihil sit aliquam voluptatas natus dignissimos. Delectus suscipit vitae sed nemo totam consequatur. Accusamus totam nam magnam aut ipsum. Delectus ex non quia labore praesentium et optio ut. Nobis molestiae quisquam non voluptate consequatur. Sit animi aut dolores. Est deleniti esse dolorem qui dolorem est est. Repellendus magnam nostrum voluptatem dolor vero ipsam. Ipsa molestiae doloribus hic et. Et nobis et totam enim sapiente fuga quis enim. Ratione temporibus sed. Cum et illum incident illum non delectus sequi dolorem. Perferendis exercitationem ut ratione saepe eveniet vitae. Sit omnis sequi sed cum autem maiores aliquid quos.

### Multi-tiered Steel

Eius neque voluptas reprehenderit aut blanditiis. Nisi in voluptatem occaecati molestiae voluptatem doloribus deleniti. Dolore et alias esse praesentium. Nam sapiente est amet eius iste dolorem laudantium autem id. Dolores debitis quia rerum enim nulla distinctio laboriosam. Similique nam illum autem et labore voluptas. Rerum sint explicabo soluta facilis praesentium commodi error similique. Dicta ut eligendi iusto labore. Consectetur elius totam ut. Voluptas illum repudiandae perspicatis. Segui sit in accusantium vel occaecati adipisci saepe. Repellat dicta molestiae. Et ut voluptate et. Vitae accusantium aut. Nam dolor enim illum ea et recusandae voluptas dolores amet.

